A CLOUD GURU

Lesson Description - Lists

In Python, there are a few different "sequence" types that we're going to work with, the most common of which being the `list` type.

Python Documentation For This Video

- Sequence Types
- Lists

Lists

A list is created in Python by using the square brackets (`[`, and `]`) and separating the values by commas. Here's an example list:

```
>>> my_list = [1, 2, 3, 4, 5]
```

There's really not a limit to how long our list can be (there is, but it's very unlikely that we'll hit it while scripting)

Reading from Lists

To access an individual element of a list you can use the index and Python uses a zero based index system.

```
>>> my_list[0]
1
>>> my_list[2]
3
```

If we try to access an index that is too high (or too low) then we'll receive an error:

```
>>> my_list[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

To make sure that we're not trying to get an index that is out of range, we can test the length using the `len` function (and then subtract 1)

```
>>> len(my_list)
5
```

Additionally, we can access subsections of a list by "slicing" it. We provide the starting index and the ending index (the object at that index won't be included).

```
>>> my_list[0:2]
[1, 2]
>>> my_list[1:]
[2, 3, 4, 5]
>>> my_list[:3]
[1, 2, 3]
>>> my_list[0::1]
[1, 2, 3, 4, 5]
>>> my_list[0::2]
[1, 3, 5]
```

Modifying a List

Unlike strings which can't be modified (you can't change a character in a string), you can change a value in a list using the subscript equals operation:

```
>>> my_list[0] = "a"
>>> my_list
['a', 2, 3, 4, 5]
```

Items in lists can be set using slices also:

```
>>> my_list[0:2] = 'a'
>>> my_list
['a', 3, 4, 5]
>>> my_list[0:2] = [1, 2, 3]
>>> my_list
[1, 2, 3, 4, 5]
>>> my_list[0:2] = []
>>> my_list
[3, 4, 5]
```

Attempting to remove an item that isn't in the list will result in an error:

```
>>> my_list.remove(6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

If the item does exist in the list then it will be removed:

```
>>> my_list.remove(4)
>>> my_list
[3, 5]
```

Items can also be removed from the end of a list using the pop method:

```
>>> my_list.pop()
5
>>> my_list
[3]
>>> my_list.pop()
3
>>> my_list
[]
>>> my_list.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: pop from empty list
```

Lists can be used to create a stack and pop can be used to take the first item like this:

```
>>> my_list = [1, 2, 3, 4]
>>> my_list.pop(0)
1
>>> my_list
[2, 3, 4]
```

Adding to the list can be done in a few ways. The first of which is by using the appe
nd method:

```
>>> my_list.append(5)
>>> my_list
[2, 3, 4, 5]
>>> my_list.insert(1, 3)
>>> my_list
[2, 3, 3, 4, 5]
>>> my_list.insert(0, 1)
>>> my_list
[1, 2, 3, 3, 4, 5]
```